From Classical Numerical Mathematics to Scientific Computing

Wolfgang Hackbusch

Abstract. The challenge of Numerical Mathematics by the fast development of the computer technology has changed this field continuously. The need of efficient algorithms is described. Their development is supported by certain principles as "hierarchical structures", and "adaptivity", "decomposition". These principles and their interactions are demonstrated in the lecture.

1991 Mathematics Subject Classification: AMS 65N, 65R, 65Y, 35A40, 45L10
Keywords and Phrases: Scientific Computing, Algorithms, pde, boundary value problems, Adaptivity, Decomposition

1 Introduction

This papers tries to sketch the structural changes in Numerical Mathematics. Due to the pages restrictions, the illustrating examples must be omitted.

1.1 The Scope of Numerical Mathematics

First, we characterise the typical topics which already appeared in Numerical Mathematics when this field developed in the mid of this century. Two essential keywords are the *approximation* (or discretisation) and the *algorithm*.

The algorithm[1] establishes the constructive part of Numerical Mathematics. In the following, we will often refer to the solution of the linear system

$$Ax = b \qquad (x, b \in \mathbb{R}^n) \tag{1}$$

as a standard example of a problem to be solved. A possible (but slow) algorithm would be the Gauß elimination performing the mapping $b \mapsto x$.

Since, in general, the mathematical problems are not solvable by finitely many elementary operations, one needs some kind of approximation. The following examples are chosen from the field of *partial differential equations* (pde). Since the solution is sought in infinitely dimensional spaces, a 'discretisation' is needed

[1]Formally, an algorithm is a function, which maps input data $x \in X$ into the desired output data $y \in Y$ and which is explicitly described by a *finite* product of *elementary* operations.

before an algorithm can be applied. The usual discretisation of a (linear) partial differential equation is a linear system (1).[2]

We obtain the following picture:

$$\boxed{\text{original problem, e.g., pde}} \tag{2a}$$
$$\downarrow \text{ discretisation process} \tag{2b}$$
$$\boxed{\text{discrete problem, e.g., system of equations}} \tag{2c}$$
$$\downarrow \text{ solution algorithm} \tag{2d}$$
$$\boxed{\text{discrete solution}} \tag{2e}$$

In the classical form of Numerical Mathematics the processes (2b) and (2d) are well separated.

Finally, the discretisation process (2b) as well as the solution algorithm (2d) are subjects of a Mathematical Analysis. The analysis of the discretisation process concerns, e.g., the discretisation error. The analysis of an algorithm may investigate its stability or its convergence speed (for iterative algorithms) etc.

1.2 Challenge by Large Scale Problems

Large scale computations are those which are almost too large to be computed on present machines.[3] Then, improvements are required to make the problem feasible. In the field of pde's it is always possible to pose larger and more complex problems than those treated at present. The increasing demands concern not only the problem dimension but also the mathematical complexity. One source of mathematical complexity is the fact that simplified models are replaced by more and more realistic ones. This may, e.g., lead to

- nonlinear problems (in the simplest case this requires a series of linear auxiliary problems to be solved, in more complicated situations the solution structure may cause further difficulties and needs respective strategies),
- complicated geometries (although the mathematical analysis of a pde for a simple two-dimensional and a complicated three-dimensional domain may be similar from a theoretical point of view, the implementation of the algorithm is by far much more involved).

Often the solution of a (discretised) pde is only a small part of the whole computation. This happens for inverse problems which may be well-posed or ill-posed. Examples are

- parameter identification problems,
- optimisation of various parameters (coefficients, shape, etc.).

[2]Another kind of approximation occurs on a lower level: the exact arithmetical operation with real number must be replaced by approximate operations in the set of machine numbers.

[3]Here, 'large scale' is to be understood in a relative sense: large compared with the computer capacity available today. In this sense, all present large scale computations will become small under future conditions.

1.3 Scientific Computing

It is the challenge by large scale problems which have changed Numerical Mathematics continuously into its present form. The changes cannot be described only by great strides made in the algorithms and in the discretisation techniques. The modern approach is characterised by a *combined* design of both, discretisations and algorithms. Even the modelling is more and more involved in the whole process. Computer Science is involved, e.g., by the modern computer architecture but also by the implementation process, which more and more becomes the bottleneck.

This paper tries to show the main strategies which have been developed and led to the present structure. In particular, we name the

- hierarchical structures,
- adaptive approaches,
- (de)composition techniques.

Hierarchies are very successful for algorithms (see §2.3), but also important for the discretisation and modelling process. Adaptive techniques are indispensable for large scale problems (see §3.2). The composition and decomposition techniques have theoretical aspects in mathematics as well as quite practical aspects as the use of parallel computers (see §4.1).

2 Efficient Algorithms

It may be self-evident that we would like the algorithms to be as efficient as possible, i.e., they should yield the desired results for lowest computational costs. This vague request can be made more precise. Below, we explain why in the case of large scale computations, the development of the computer technology leads to the need of algorithms with linear complexity. The notation of complexity is recalled below.

2.1 Algorithms and Their Complexity

In the following, we fix the discretisation (2b) and discuss the algorithm (2d).

While the structural properties of algorithms are quite similar to those of proofs in mathematics, two algorithms $\alpha, \beta : X \to Y$ mapping the input x into the same output $\alpha(x) = \beta(x)$ are not considered to be equal but are valuated according to their costs. Typical cost criteria are the required computer time and storage. Since the time needed for the computation depends on the speed of the computer, we may take the number of elementary arithmetical operations as a measure.[4] Since, by definition, each algorithm $\alpha : X \to Y$ is a well-defined product $\alpha = \alpha_k \circ \ldots \circ \alpha_1$ of elementary operations α_i, the arithmetical costs $C(\alpha)$ of an algorithm is well-defined, too.

Usually, the data sets X, Y are not fixed but can be parametrised (e.g., $X = R^n$). Let n be the maximum of the number of input and output data. The complexity of an algorithm α is $O(\varphi(n))$, if $C(\alpha) = O(\varphi(n))$ as $n \to \infty$.

[4]This is a simplifying assumption. In fact, on modern computers the relation between the number of arithmetical operations and the computer time is no more linear, e.g., because of pipelining effects.

There are difficult problems, for which it is considered as a success if the complexity is polynomial (i.e., $\varphi(n) = n^p$ for some p). To this respect, problems from Linear Algebra are simple. For instance, the $n \times n$ system (1) can be solved by Gauß elimination with complexity $O(n^3)$. But as we will see below, the $O(n^3)$ complexity is quite unsatisfactory.

Since, except trivial counter-examples, n data require at least one operation per datum, the *linear complexity* $O(n)$ is the best possible (as a lower bound). Whether linear complexity can be achieved is often an open problem.

Instead of the polynomial complexity behaviour $O(n^p)$, one often finds the asymptotic behaviour[5] $O(n^p \log^q n)$. Because of the slow increase[6] of the logarithm, the logarithmic factor is considered as less important. We say that the complexity is *almost linear* if $p = 1$, while $q > 0$ is allowed.

To simplify the discussion, we have concentrated on the number of arithmetical operations (computer time) and have not mentioned the storage requirements. If nothing else is said, we suppose that the storage requirement is (almost) linear in n.

2.2 Why Linear Complexity is Necessary

The asymptotic description of the algorithmic complexity is uninteresting as long as we are not forced to increase n. This need is caused by the computer technology. In the former times of hand calculations or mechanical calculators, there were obvious reasons why n was rather small. This is why Numerical Mathematics did not appear as a discipline of its own before the help of electronic computers was available.

As pointed out in §1.2, we would like to compute problems as large as the computer resources allow. Assuming a storage requirement of $O(n)$, we conclude that the dimension n of the largest problem we can handle increases directly with the storage of the computer.

The steady improvement of the computer technology can be described quantitatively. In spite of the technological jumps, the improvement of the storage size is rather uniform over the past decades. One observes an improvement by a factor about 100 over 10 years. A similar factor can be found for the increase in speed. The only interesting fact from these data is that *storage and speed increase by almost the same factor per time.* This has an immediate impact on the computer time for the problems to be solved.

Suppose an algorithm with complexity $O(n^p) \approx Cn^p$. Replacing the old computer by a new one with storage and speed improved by the factor $c > 1$, we want to solve problems of dimension cn instead of n (due to the increased storage). This requires $C(cn)^p$ operations. Because of the improved speed, the computer time is now $C(cn)^p/c = Cc^{p-1}n^p$ instead of Cn^p previously. We conclude that an improvement of the computer facilities by c increases the computer time by c^{p-1}. Hence, only if the algorithm has (almost) linear complexity, the run time does not deteriorate.

[5]Or more general $O(n^{q+\varepsilon})$ for any $\varepsilon > 0$.

[6]In fact, the constants in two $O(n^p \log^q n)$ terms can be more important than the logarithm.

The conclusion for algorithms with complexities worse than the linear one is that either the algorithm can only be used for small size problems or one has to tolerate larger and larger computational times.

2.3 Hierarchical Structures

One basic principle that may lead to efficient algorithms it the use of hierarchies. A typical advantage of a hierarchical structure is the possibility of recursive algorithms. Below, we give a well-known example.

2.3.1 Example: FFT

Consider Eq. (1) with matrix entries $a_{jk} = \omega^{jk}$ $(0 \leq j, k \leq n-1)$, where $\omega = \exp(\pm 2\pi i/n)$. Then the matrix-vector multiplication $x \longmapsto b = Ax$ describes the mapping from the vector coefficients x into the Fourier coefficients of $b = \mathcal{F}_n(x)$ (or vice versa, depending on the $\pm$ sign).

The standard matrix-vector multiplication algorithm has $O(n^2)$ complexity. Let $n = 2^q$. The idea of the Fast Fourier Transform (FFT), which can be traced back to Gauß, is to split the unknown Fourier coefficients $b = (b_0, b_1, \ldots, b_{n-1})$ into $b_{odd} = (b_1, b_3, \ldots, b_{n-1})$ and $b_{even} = (b_0, b_2, \ldots, b_{n-2})$ and to construct the related x_{odd}, x_{even} with $b_{odd} = \mathcal{F}_{n/2}(x_{odd})$, $b_{even} = \mathcal{F}_{n/2}(x_{even})$. This allows a recursive application: One problem of dimension $n = 2^q$ (level q) is transferred into 2 problems of dimension $n/2 = 2^{q-1}$ (level $q-1$), etc. until it is reduced to $n = 2^q$ problems of the trivial dimension 1 (level 0). The costs per step are $O(n)$. Since $q = \log_2 n$ is the number of levels, we result in the almost linear complexity $O(n \log n)$.

Here, the vector spaces $X_\ell = \mathbb{R}^{n_\ell}$ of dimension $n_\ell = 2^\ell$ $(\ell = 0, 1, \ldots)$ form the hierarchy. The typical characteristics of the FFT algorithm are: (i) The problem is trivial at level 0, while (ii) it is easy (and cheap) to reformulate the problem of level ℓ by those of level $\ell - 1$. In more general cases, (ii) takes the form that an essential part of the algorithm is the solution of problems on the lower level.

2.3.2 Example: Wavelets

The fact that the number of involved hierarchy levels grows like $\log_2 n$ does not necessarily imply that this logarithmical factor must appear in the complexity. The wavelet transformation, which is quite close to the Fourier Transform, relies much stronger on the hierarchical structure (functions f of level[7] ℓ define functions $f(2\cdot)$ of level $\ell + 1$ and vice versa). Supposing a finite filter length, the wavelet transform and its back transform have exactly *linear* complexity.

The hierarchy for wavelets defined on $\mathbb{R}$ is the family $\{G_\ell = \{x = k2^{-\ell} : k \in \mathbb{Z}\} : \ell \in \mathbb{Z}\}$ of uniform grids. Since the wavelets are a part of Mathematical Analysis and a tool for the approximation, we see that the concept of hierarchies is also essential for the discretisation process.

[7] In wavelet terminology 'level' is called 'scale'.

2.3.3 Example: Solution of Sparse Systems by Multi-Grid

Most of the discretisation methods for pde produce a so-called *sparse* matrix A in (1), i.e., the number of non-zero entries is much smaller than n^2; in the following, we assume that there are $O(n)$ non-zero entries. A trivial consequence is that the matrix-vector multiplication $x \mapsto Ax$ is cheap (linear complexity). Therefore, the hope is to approximate[8] the solution by an iterative process using only a fixed (n-independent) number of such matrix-vector multiplications.

Although A is sparse, the inverse A^{-1} is, in general, a full matrix. This allows the following illustration of the difficulty about linear complexity. Even if we would be able to get the inverse matrix A^{-1} for free, the computation of x by $x := A^{-1}b$ involves the multiplication of a full matrix by a vector and is therefore of complexity $O(n^2)$.

Linear iterations for solving $Ax = b$ are of the form[9]

$$x^{m+1} = \Phi(x^m, b) := Mx^m + Nb = x^m - N(Ax^m - b)$$

with the iteration matrix $M = I - NA$ (N arbitrary). The iteration converges, $x^m \to x = A^{-1}b$, if the convergence speed which equals the spectral radius $\rho(M)$ of the matrix M is < 1. In order to get the best results for minimal costs, one has to minimise the *effective work*

$$Eff(\Phi) := \frac{\text{cost per iteration step}}{-\log \rho(M)} = \min$$

over all linear iterations Φ. It turns out that Φ leads to an almost $O(n^p)$ complexity for the solution of (1) if $Eff(\Phi) = O(n^p)$. Due to the sparsity, we may assume 'cost per iteration step'$= O(n)$; hence, $Eff(\Phi) = O(n^p)$ is equivalent to $\rho(M) = 1 - O(n^{1-p})$. In particular, linear complexity requires $\rho(M) \le \bar{\rho} < 1$ for all n.

Unfortunately, there is no iteration known so far which ensures linear complexity for *all* sparse matrices A. Instead one looks for fast iterations that work for *certain classes of matrices*.

Such a class are the sparse matrices resulting from the discretisation of elliptic partial differential equations, where the *multi-grid iteration* leads to linear complexity. The characteristic structure of the multi-grid method is the use of a hierarchy of discrete problems. The standard hierarchy parameter is the grid size h. Denote the discrete problem on hierarchy level ℓ by $A_\ell x_\ell = b_\ell$ for decreasing mesh sizes $h_0 > h_1 > \ldots > h_\ell > \ldots$. The iteration for solving a discrete problem of level ℓ involves the lower levels $0, 1, \ldots, \ell - 1$ as auxiliary problems. A brief explanation of the fast convergence is as follows: Standard classical iterations have a local range and reduce very well the oscillatory iteration errors. Long range errors need long range corrections which can be performed efficiently

[8]There is no need to compute the discrete solution too accurate, since we are interested in the solution of the problem (2a). The discrete solution is affected with the discretisation error in any way. Hence, an additional approximation error of the size of the discretisation error is acceptable.

[9]For details see Hackbusch: Iterative solution of large sparse systems of equations. Springer, New York 1994.

only by coarser grids corresponding to lower levels. Algebraic properties of A like positive definiteness, symmetry, etc. are less important[10]. The main properties needed in the convergence proof is the fact that the family of matrices A_ℓ stems from a discretisation of an elliptic pde.

2.3.4 Difficulties due to Complicated Geometries

Large scale problems involve possibly an increasingly detailed geometry, since now more and more data are available for the geometric description. While technical objects have a comparably simple shape, problems from medicine or geography etc. may be rather complicated.

We recall that the multi-grid method requires a hierarchy of grids of size h_ℓ starting with a quite coarse grid size h_0. Although these grids can be constructed by the very flexible finite elements, the existence of such a grid hierarchy seems to be in conflict with a detailed geometry, since a complicated geometry requires that all describing grids are small enough. Here, a progress can be reported. Independent of the smallness of the geometrical details, one can construct a hierarchy of nested (conforming) finite element spaces

$$V_0 \subset V_1 \subset \ldots \subset V_{\ell-1} \subset V_\ell \subset \ldots \subset H^1(\Omega)$$

(so-called *composite finite elements*[11]) so that $\dim V_0$ can be a small number (equivalently, the corresponding mesh size h_0 can be rather larger, e.g., h_0 can be of the diameter of the domain). Although the size h_ℓ may be much larger than the size of the geometrical details, one can prove the standard approximation $\inf\{\|u-u_\ell\|_{H^1(\Omega)} : u_\ell \in V_\ell\} \leq C h_\ell \|u\|_{H^2(\Omega)}$ for all $u \in H^2(\Omega)$, which is fundamental for the error estimation and multi-grid convergence.

2.4 Robustness versus Efficiency

The example of the multi-grid method has shown that, in order to obtain efficiency, one has to make use of the special properties of the considered subclass of problems. In the case of multi-grid, the strength of ellipticity is one of these properties. In singular perturbation problems, ellipticity is fading out. Furthermore, there are other problem parameters which can have a negative influence on the convergence speed of the iteration. As soon as convergence can turn into divergence, the method becomes unreliable.

We call an algorithm *robust* (with respect to a certain set and range of parameters) if its performance does not fail when the problem parameters vary. Often,

[10]This is underlined by the fact that even *nonlinear* systems can be solved by (nonlinear) multi-grid iterations with asymptotically the same speed.

[11]Details in a) Hackbusch, Sauter: Composite finite elements for the approximation of PDEs on domains with complicated mirco-structures. *Numer. Math.* 75 (1997) 447-472; b) Hackbusch, Sauter: Composite finite elements for problems containing small geometrical details. Part II: Implementation and numerical results. *Computing and Vizualization in Science* 1 (1997) 15-25; c) Sauter: Composite finite elements for problems with complicated boundary. Part III: Essential boundary conditions. Report 97-16, Universität zu Kiel.

one has to find a compromise between quite efficient but non-robust and very robust but inefficient methods. There are various approaches to robust multi-grid variants, e.g., the 'algebraic multi-grid method'. The term 'algebraic' indicates that the method uses only the information of the algebraic data in (1) and does not require details about the underlying pde and the discretisation process. Such a method comes closer to a 'black-box method', but is has to be emphasised that the algebraic multi-grid methods are still restricted to a subclass of systems.[12]

The preference for robust or for very efficient but more specialised methods also depends on the kind of user. While the numerical mathematician likes highly efficient algorithms for a special application, other users prefer robust methods since either the mathematical background is not well-understood or not available.

3 Efficient Discretisation Methods

It is not enough that the solution method is efficient. Also the discretisation of the partial differential equation must be considered. In academic situations, the order of the discretisation is essential and new kinds of approximations can be proposed (see next Subsection). Nevertheless, in general, one needs adaptive methods. The reasons for adaptivity and the tools for its implementation are considered in §3.2.

3.1 Comparison of Different Discretisation Methods

So far, we have taken the discrete problem as given and were looking for an efficient solution algorithm. Hence, the discretisation process in (2b) was considered to be fixed. Instead, one should also compare different discretisation methods. The success of an discretisation can be judged by the *discretisation error*, the difference[13] between the exact and approximate solution, or a suitable norm of the discretisation error. Here, it is to be emphasised that the discretisation method does not produce only one particular discrete problem, but at least a sequence (or as we shall see later, even a larger set) of discrete problems. Using the dimension n as an index, we may write the discretisation method $\mathcal{D}$ as the sequence $(P_n)_{n\in\mathbb{N}'\subset\mathbb{N}}$ of discrete problems P_n with solution x_n and discretisation error ε_n.

Usually, the aim is to reach the best accuracy for minimal costs. To be more precise, two particular strategies are of interest:

- *Accuracy oriented choice:* Let an accuracy $\varepsilon > 0$ be given. For a fixed discretisation choose the minimal dimension $n = n_\varepsilon$ such that the discretisation error is $\precsim \varepsilon$. The arithmetical costs are denoted by Costs(P_{n_ε}). Choose that discretisation method for which Costs(P_{n_ε}) is minimal.[14]

[12]The scope of the method is not easy to describe, since one observes that it performs well even for situations where convergence proofs are still missing.

[13]The definition of this 'difference' is not quite unique since the discrete and the continuous solution are elements from different sets.

[14]If the discrete problems of the different discretisations are of the same kind (hence, the costs depend only on n), the discretisation with minimal n_ε is sought.

- *Memory oriented choice:* Let a maximal data size N be given (e.g., the whole memory of the computer). Choose that discretisation method which yields a discrete solution x_N (for the particular N) with best accuracy ε_N.

The accuracy oriented choice is the more advanced one. The difficulty in practice is twofold. The first difficulty concerns the prediction of ε. Often it is not easy to tell how accurate (with respect to what – global or weighted – norm) the solution should be. Second, it is not trivial to judge the error of the discrete solution, i.e., to check whether *error* $\precsim \varepsilon$.

The memory oriented choice is a lazy choice. The whole computer capacity may be used although the result will be much too accurate for the purpose in mind.

These alternatives can be illustrated for two discretisation methods of different order. Let $\mathcal{D}_I$ be a first discretisation of order α, i.e., the discretisation error $\varepsilon_{I,N}$ behaves like[15] $O(n^{-\alpha})$, when the dimension n varies. Similarly, let $\mathcal{D}_{II}$ be a second discretisation method of order β. For the accuracy oriented choice, $\varepsilon = O(n_I^{-\alpha}) = O(n_{II}^{-\beta})$ yields $n_I = O(\varepsilon^{-1/\alpha})$ and $n_{II} = O(\varepsilon^{-1/\beta})$. Hence, $\alpha < \beta$ implies that (at least asymptotically) $n_{II} < n_I$ and therefore the higher order discretisation is more efficient. For the memory oriented choice, $n = N$ is fixed. Again, the higher order $\beta > \alpha$ is preferred, since the accuracy $\varepsilon_I = O(N^{-\beta})$ is (asymptotically) smaller than $\varepsilon_{II} = O(N^{-\alpha})$.

Attempts have be made to improve the polynomial behaviour $\varepsilon = O(n^{-p})$. One approach is the *p-finite element method*, where the step size h remains fixed, while the order $p = p(n)$ is increasing. Under perfect conditions, an exponential behaviour $\varepsilon = O(\exp(-cn^{\alpha})$ $(c, \alpha > 0)$ is obtained.[16]

Another approach are the *sparse grids*, where the discretisation error is almost of the order $\varepsilon_h = O(h^p)$, whereas the grid has only $n = O(h^{-1})$ grid points even if the domain is a subset of $\mathbb{R}^d$. Then, the discretisation error equals $\varepsilon_h = O(n^{-p})$ instead of $O(n^{-p/d})$. Since $d = 3$ is the standard spatial dimension, this approach promises a much better accuracy for the same dimension n.

In practice, both of the methods mentioned above cannot be applied to general boundary value problems, but only to local parts. In the case of the p-method, the solution must be very smooth, which may happen in the interior of the domain with a fixed distance from the boundary but is in general not true at the boundary. This gives rise to the *hp-method* which combines the standard finite element method with the p-method in an adaptive manner. In the case of sparse grids, these grids correspond to a special domain (square, cube etc. or their smooth image), which is usually only a part of the whole domain. Therefore, in general, the use of p- and sparse-grid methods require in addition adaptive techniques as they are explained below.

[15]This definition is simplified. Usually the order is defined by $O(h^{-\alpha})$, where h is the mesh size. h and n are connected by $n = O(h^{-d})$, where d is the dimension of the domain $\Omega \subset \mathbb{R}^d$.

[16]To be precise, one has also to take into account that the p-method requires much more accurate quadratures for the system matrix entries and that the resulting linear system is harder to solve than standard finite element systems.

3.2 Adaptivity

3.2.1 Abstract Setting

To be precise, the result of a discretisation is a *family* of discrete problems $\mathcal{P} = \{P_i : i \in I\}$, where the index set I usually coincides with $\mathbb{N}$ or an infinite subset of $\mathbb{N}$. If x_i is the solution of P_i, we expect a certain of convergence of x_i to the solution x of the continuous problem, i.e., the discretisation error ε_i should tend to 0. In the case of the 'accuracy oriented choice' from §3.1, we want to find the minimiser $P_{i_{opt}}$ of $\min\{\mathrm{costs}(P_i) : i \in I \text{ and } \varepsilon_i \leq \varepsilon\}$. The trivial strategy for finding i_{opt} is to test the solution x_i and to proceed to index $i+1$ (this can, e.g., mean a halving of the mesh size) if $\varepsilon_i > \varepsilon$.

In the adaptive case, the index set I has a much more general structure, e.g., it may be a graph. Then, given a discrete problem P_i, there are several next finer discrete problems $\{p_j : j \text{ successors of } i\}$. The solution of the minimisation problem $\min\{\mathrm{Costs}(P_i) : i \in I \text{ and } \varepsilon_i \leq \varepsilon\}$ must be avoided.[17] Instead, one needs a heuristic H selecting a convergent subsequence $\{P_{i_k} : k \in \mathbb{N}\}$, $i_k = H(x_{i_{k-1}})$.

If, in the Galerkin case, adaptation is understood more generally as the optimal approximation by any kind of function spaces, the theoretical background traces back to the *n-widths* introduced by Kolmogorov.

3.2.2 What Parameters can be Adapted?

The finite element discretisation decomposes the whole domain into triangles (tetrahedra) or other geometric elements. Starting with a given (coarse) finite element triangulation of a domain with step size h_0, we can consider a uniform refinement (e.g., each triangle is regularly divided into four smaller ones). This yields a sequence of discrete problems with the uniform step size $h_\ell = 2^{-\ell}h_0$. On the other hand, the finite element discretisation allows to choose *different* element sizes at different locations, i.e., the mesh size may become a function $h(x)$. Among all finite element discretisations one has to select a sequence satisfying $\lim \max_x h(x) = 0$. Usually, the triangulations τ_i of this sequence are not chosen independently, but given a triangulation τ_i the next one, τ_{i+1}, is obtained by *local* refinement. The question arises where to refine the grid.

The adaptation by local grid refinement is the most important example which we shall discuss below. For completeness, also other subjects of adaptation are mentioned. (i) Another possibility is to adapt the order of the finite element functions (hp-method). (ii) Usually, one avoids flat (almost degenerated) triangles. However, under certain circumstances, flat triangles with a prescribed direction of the longest side are desired. Therefore, the orientation and degree of degeneration is a possible subject of adaptation. (iii) The kind of discretisation technique may change in different subregions of the boundary value problem.

[17] The minimisation over certain discretisation parameters is a problem of a much higher complexity than the original task. Hence, the final costs is not $\mathrm{Costs}(p_i)$ for a suitable i, but $\mathrm{Costs}(p_i)$ plus a large overhead for the minimisation.

3.2.3 Why should be Adapted?

A uniform step size is (almost) optimal, if the function to be approximated is uniformly smooth. In practice, one has to approximate functions with different smoothness in different parts, e.g., (i) very smooth in one part $\Omega_{(i)}$ of the domain and (ii) less smooth in another part $\Omega_{(ii)}$. Then the mesh size might be constant $(= h^*_{(i)})$ in $\Omega_{(i)}$ and constant $(= h^*_{(ii)})$ in $\Omega_{(ii)}$ but with $h^*_{(i)} \gg h^*_{(ii)}$. Choosing the uniform but coarse mesh size $h = h^*_{(i)}$ everywhere, we result in a large discretisation error because of the bad approximation in $\Omega_{(ii)}$. On the other hand, the uniform choice $h = h^*_{(ii)}$ gives (by definition) a satisfactory discretisation error, but because of $h^*_{(ii)} \ll h^*_{(i)}$, this grid is much too fine in $\Omega_{(i)}$ and leads to a total dimension much larger than necessary.

Often, a further situation arises: (iii) The function has singular derivatives at a certain point x_0. Then, one needs a mesh with $h^*(x)$ decaying in a certain way as x approaches x_0. Note that $h^*(x)$ takes very small values only in very smalls parts of the domain. Choosing such a fine grid everywhere would be a huge waste of computer time.

Altogether, one has to construct a mesh with local mesh width $h(x) \leq h^*_{(i)}, h^*_{(ii)}, h^*(x)$ in the respective parts.

3.2.4 What makes Adaptation Difficult?

The reason for the different choices of $h(x)$ is the smoothness of the function $u(x)$ to be approximated. In simple cases like quadrature, the function u and possibly its derivatives are explicitly available. A different situation occurs in the case of differential equations. Here, the function u is the quantity we are looking for. The question arises whether we can get the information about the smoothness of u before we have computed the approximation of u.

The answer to the latter question is that an iterative approach is used. Starting with a rough approximation of u^0, one tries to find informations for adapting the mesh from which the next approximation u^1 is computed, etc.

This iteration combines the discretisation process and the solution process, since they are performed in a cyclic manner.

3.2.5 How to Control the Adaptation?

There are cases, where the adaptation to the problem can be designed *a priori*, but, usually, the adaptation process is done *a posteriori*, more precisely, during the computational process. For the a posteriori adaptation, we have to describe the control mechanism steering the details of the adaptation.

A general strategy to this respect consists of two fundamental considerations:

- The discretisation error is to be described as a sum of *local errors*. Usually, the local residuum is such a tool.
- The desired situation is the *equidistribution* of the local errors. That means, one tries to adapt the mesh so that all local errors are equally sized. The

argument is that a locally small error is a waste of computation without improving the global error essentially.

The control mechanism is first explained for the 'memory oriented choice' explained in §3.1. In this case, one refines as long as further storage is available. The only critical decision is where to refine the finite element grid. For this purpose, a lot of 'error indicators' exist which indicate where (possibly) the error is dominating. Many of these criteria are heuristic. The theory-based error estimators are explained below.

The 'accuracy oriented choice' from §3.1 requires two decisions. First, we need an indication that the discretisation error is below the required accuracy ε. In the negative case, we have to decide where to refine locally (as discussed above). Both decisions are supported by the error estimators explained in the next Subsection.

In particular in time dependent problems, not only an adaptive refinement but also a *coarsening* may be necessary.

3.3 Error Indicators and Estimators

The *a posteriori error estimators*, first introduced by Babuška and Rheinboldt[18], are a fundamental tool for the adaptive refinement. Let τ be a triangulation of the domain Ω, i.e., Ω is the disjoint union of the elements $\Delta \in \tau$. The finite element solution for the triangulation τ is denoted by u_τ. Then, the error estimator has the form

$$\Phi(u_\tau) = \sqrt{\textstyle\sum_{\Delta\in\tau}\varphi_\Delta(u_\tau)},$$

where φ_Δ is a computable[19] function depending only on the data restricted to Δ (or its neighbourhood). Denoting the error of u_τ by $e(u_\tau)$ (e.g., $e(u_\tau) = \|u - u_\tau\|$ for a suitable norm $\|\cdot\|$), we would like to have constants A, B such that

$$A\Phi(u_\tau) \le e(u_\tau) \le B\Phi(u_\tau).$$

If $e(u_\tau) \le B\Phi(u_\tau)$ holds, Φ is called *reliable* since knowing its value we can guarantee an error estimate. If $A\Phi(u_\tau) \le e(u_\tau)$, Φ is called *efficient* since we avoid overestimation.

3.4 Combination of the Discretisation and Solution Process

In the beginning, we said that in the classical form of Numerical Mathematics the discretisation of the continuous problem and the algorithm for the discrete

[18]See Babuška-Rheinboldt: A posteriori error estimates for the finite element method. *Int. J. Numer. Meth. Engrg.* 12 (1978) 1597-1615. For a recent survey see Verfürth: A review of a posteriori error estimation and adaptive mesh-refinement techniques. Wiley-Teubner 1996.

[19]To be quite precise, there are two alternatives to be considered. 1) If φ_Δ is a mathematical expression including integration, we can obtain reliable error estimates. 2) For computational purposes, such a φ_Δ (e.g., the integration contained in φ_Δ) must be discretised and yields an algorithm $\tilde\varphi_\Delta$. Then, Φ cannot be reliable in general without (a priori) assumptions on the smoothness of the integrands.

problem were well separated. With the adaptive approach we have reached a new level, where a new kind of algorithm is *directly applied to the continuous problem*, i.e., the design of the discretisation has become a part of the solution algorithm itself.

The reason for this development is not only the efficacy we want to obtain, but also the huge amount of data. As long as we compute only few numbers, we may be able to judge their quality and possibly improve the discretisation. However, when we compute a massive set of data corresponding, e.g., to a relative dense three-dimensional grid, we have already problems to perceive the data. We need special visualisation tools to interpret the computed results. The judgement of their accuracy is even more difficult. Therefore, it is an obvious consequence that the control over the discretisation process is given to the algorithm itself.

The new kind of algorithm can be considered as a triple $(\mathcal{D}, \mathcal{A}, \mathcal{H})$, where $\mathcal{D}$ is the discretisation method (offering a large variety of discrete problems, e.g., all finite element triangulations), $\mathcal{A}$ are the algorithms for solving the discrete problems produced by $\mathcal{D}$, while the heuristic $\mathcal{H}$ is the adaptive strategy controlling the discretisation process.

3.5 Hierarchy plus Adaptivity

In the following, we discuss the hierarchy of grids used by the multi-grid method. Then adaptive approaches can be realised in two ways.

1) *Global grids.* Let $\{G_\ell : \ell = 0, 1, \dots\}$ be the sequence of grids (finite element meshes etc.), where $G_{\ell+1}$ is constructed adaptively from the solution x_ℓ in grid G_ℓ.

2) *Local grids.* Let $G'_0 = G_0$ be a starting grid and denote by G'_ℓ the regular refinement (ℓ partitioning steps in all elements). An adaptive (local) refinement G_1 of G_0 can be considered as a union of $G_{10} := G'_0$ and of a subset $G_{11} \subset G'_1$. In general, a local refinement G_ℓ is a union of subsets $G_{\ell,k} \subset G'_k$ $(0 \le k \le \ell)$.

The second approach works also for the wavelet hierarchy: There the local refinement is replaced by adding the wavelet functions $\psi(2^\ell x - k)$ of level ℓ for only few shifts k.

So far, the hierarchical structure does allow adaptivity. Of course, extra overhead occurs to administrate the additional description of the local grid details.

4 Parallelism

The costs of an algorithm are not determined by mathematics but by kind of computing tools. If the technology is changing also the valuation of algorithms might change. For instance, on a vector computers Costs(A_I)<Costs(A_{II}) may hold, although algorithm A_I requires more scalar operations than A_{II}, provided that A_I exploits the vector operations.

In the last decade, the parallel computer became available which allows to perform the computation in parallel on a number of processors, provided the computations are independent. In the optimal case (optimal balance, no overhead) the computation time decreases by the factor p=number of processors. Another effect is the enlarged storage (p times the storage of each processor), provided that the algorithm can use the distributed memory. Since, in the optimal case, speed and storage increases by the same factor, the considerations of §2.1 show that also the parallel algorithms must be of linear complexity.

Let a sequential algorithm A_s be given. One can try to construct a parallel algorithm A_p which yields identical results. For its construction, one needs at least a *data decomposition.*

Usually, one tries to construct a special parallel algorithm. One strategy for its construction is the *problem decomposition* of the full problem into subtasks.

4.1 Composition, Decomposition

4.1.1 Composition

Often, large scale problem are obtained by composing subproblems. Difficulties in the decomposition process may possibly arise from

a) different kinds of differential equations and/or integral equations in the subproblems,

b) different coordinate systems in the subproblems,

c) different discretisations in the subproblems,

d) non-fitting meshes even when all subproblems are discretised by the same kind of finite elements.

The coupling conditions, which are similar to the boundary conditions, must be integrated into the complete problem. If the meshes do not fit, one has to ensure the connection in a weak sense, e.g., by Lagrange multipliers (so-called 'mortar element method').

4.1.2 Decomposition

The decomposition of the whole problem into subproblems can have different reasons:

1) software is available for the specific subproblems,

2) the iterative scheme makes use of the solution of the subproblems,

3) the problem must be decomposed to use a parallel computer.

Another question is how the complete problem can be divided. Two different approaches are relevant:

a) The given problem is already a composed problem, then the obvious candidates for the subproblems are the basic components.

b) If the given problem is uniform, a partitioning must be defined. Differently from a), the number of subtasks can be chosen according to the number of available processors.

Reason 1) is, in particular, important for large scale problems which are implemented by a team where each expert is responsible for a particular subtask.

Reason 2): For iterative schemes[20], it is a standard approach to correct the

[20]Details in Chapter 11 of Hackbusch: Iterative solution of large sparse systems of equations. Springer, New York 1994.

actual approximation by a solution of a simpler problem[21], where the 'simpler problem' is obtained by neglecting the coupling of subproblems. The simplest subproblems of a system (1) are the separate n scalar equations. Solving the ith scalar equations with respect to x_i yields the classical Jacobi and Gauß-Seidel iteration. Since a partitioning according to Approach a) is obvious, we consider the Approach b). In the context of elliptic pde's discretised over a mesh in the domain Ω, one can partition Ω into subdomains Ω_i (together with their meshes) such that $\cup\bar{\Omega}_i = \bar{\Omega}$. This leads to the *domain decomposition method*. The decomposition may also use overlapping domains. Since the first domain decomposition method (with two overlapping domains) was used by H. A. Schwarz (1870) to prove the existence of a holomorphic function in a composed domain, these iterations are also called *Schwarz iteration*.

The use of parallel computers for domain decomposition methods is obvious: The solution of the ith subproblem on Ω_i involves intensive computations on the ith processor. Afterwards communication is needed to initialise the next iteration step, but the communication concerns only the overlapping region or in the simplest case only the common interior boundary. Since the communication involves only a rather small part of all unknowns, there is a hope for a good speed-up factor.

However, the use of the domain decomposition principle *only* cannot be successful. If p is the number of subdomains (and parallel processors), the overlapping Schwarz method does lead to a speed-up by p, but the convergence speed of the iteration slows down by the same factor. Therefore, in the meantime it is well-accepted that one has to add a coarse-grid subspace.[22] This makes the domain decomposition approach very similar to the multi-grid method: The coarse-grid correction has a larger step size ratio h_{fine}/h_{coarse}, while the subspace solutions form the smoothing process of the two-grid iteration.

The addition of the coarse-grid subspace leads to a generalisation of the domain decomposition principle: The decomposition of the vector space into subspaces. The resulting notation of a *subspace iteration* is general enough to describe the domain decomposition methods as well as the multi-grid iterations. The theory developed so far[23] is more or less restricted to positive definite system matrices A. Applied to multi-grid iterations, the results use weaker assumptions but yield also weaker convergence results.

4.2 Interaction of these Principles

4.2.1 Hierarchy plus Decomposition

The hierarchy can be considered as a vertical structure providing problems of different discretisation levels, whereas the decomposition yields an horizontal structure.

[21]Let W be 'close' to A but such that $Wy = d$ is easy to solve. Then the iterative scheme $x^{new} = x^{old} - W^{-1}(Ax^{old} - b)$ requires the solution of $Wy = d$ with $d = Ax^{old} - b$.

[22]Divide the domain Ω into pieces Ω_i of size H and introduce a global mesh of size h. Then the coarse-grid mesh has size H.

[23]Survey in Xu: Iterative methods by space decompositions and subspace correction. *SIAM Review* 34 (1992) 581-613.

These structures are essentially orthogonal and do not conflict with each other.

The traditional domain decomposition method has two hierarchy levels, the global problem and many local subproblems. It is possible to repeat the domain decomposition principle for each subproblem. Another possibility is to use the same decomposition structure over all levels on the hierarchy. This is the standard approach for *data decomposition* for the purpose of parallel computations.

Hierarchy plus parallelism may create a specific problem. Usually, the algorithm works sequentially over the hierarchy levels. If the lower levels are connected with coarser grids and therefore less computational work, the communication part may predominate.

4.2.2 Decomposition plus Adaptivity

When using the decomposition for parallelising, the idea is to associate each subproblem with one of the processors. At the starting time of each iteration step, all processors must get the new (boundary and right-hand side) data for the subproblems. Since the iteration cannot proceed before all results are collected, one should ensure that all subtask computations need almost the same time. This requirement can be satisfied by creating subdomains with nearly the same number of unknowns.

In this case, adaptivity leads to a severe conflict. By definition, the adaptive refinement yields locally different changes. One subdomain may be strongly refined, whereas another one remains unchanged. Obviously, even if the dimensions of the subtasks are equidistributed initially, the subproblems may lose their balance. Without a rearrangement of the subdomains, the parallel algorithm becomes poor.

The rearrangement process is called *load balancing*. On the one hand side, the load balancing must be cheap in order not to spoil the overall performance time. On the other hand, the load balancing is a very delicate task because a) the optimal decomposition is NP-hard, b) the subdomain data to be rearranged on one processor are distributed over different processors. It becomes even more difficult in the multi-grid case where also the vertical level structure is to be considered.[24]

If the load balancing is successfully implemented, the algorithm decides not only about the termination (when the accuracy is reached) and local refinement, but also about the decomposition structure.

5 Modelling and Implementational Aspects

5.1 Modelling

We started with an approximation (discretisation) separated from the algorithm for the discrete problem. As shown in §3.4, both have become more and more intertwined. However, the mathematical problem from (2a) is not really fixed. Usually, it is the result of a modelling process for some problem from outside

[24]See Bastian: Parallele adaptive Mehrgitterverfahren. Teubner, Stuttgart 1996.

mathematics (e.g., mechanics; see (3a)). The modelling process may be an approximation[25] by itself. The model might be more or less involved, certain aspects may be neglected or simplified or represented in full detail. Often, the details of the modelling process should be related to the accuracy required for the discrete problem. This gives rise to a hierarchy of models.

$$\boxed{\text{physical (chemical etc.) problem}} \tag{3a}$$

$$\downarrow \text{ modelling process} \tag{3b}$$

$$\boxed{\text{mathematical problem, e.g., pde}} \tag{3c}$$

$$\downarrow \text{ discretisation process} \tag{3d}$$

$$\boxed{\text{discrete problem, e.g., system of equations}} \tag{3e}$$

$$\boxed{\text{computer} \longleftrightarrow} \downarrow \text{ solution algorithm} \tag{3f}$$

$$\boxed{\text{discrete solution}} \tag{3g}$$

From the mathematical point of view it is very interesting when the model hierarchy leads to *different scales* in the solution. Such different scales may be *time scales* for time-dependent problem: Certain processes are much faster than others (e.g., mechanical changes faster than thermal ones or chemical reactions faster than the flow dynamics). This gives rise to interesting discretisation techniques. The consideration of scales in the discretisation can also be regarded as an adaptation process (using the smallest time scale for *all* components would be a waste of computer time).

Details in a model may also lead to *geometric scales*. The diameter of the domain (of the boundary value problem $Lu = f$) is the coarsest scale. The coefficient function of L may be oscillatory giving rise to the wavelength as next geometric scale. In regular cases, the *homogenisation technique* offers a tool to split the true solution into a sum of a homogenised part and the details.

5.2 Implementation

In (3f) the box 'computer' should indicate the interaction of the solution algorithm with the computer. This includes that the algorithm depends on the computer architecture. Another important software aspect is mentioned next.

The steadily increasing volume of the data and the increasing problem complexity on the one hand and the development in the computer architecture on the other hand have made the implementation more and more involved. Although algorithms and computers have become faster, the act of implementation consumes an increasing time of work. Since Scientific Computing needs extensive software, its production (i.e., the implementation process) must become a scientific topic of Scientific Computing by its own.[26]

[25] This approximation process is meant when engineers speak about a *simulation*.

[26] For a positive example see Bastian et al.: UG - A flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science* 1 (1997) 27-40.

6 Treatment of Non-Sparse Matrices

The request of linear complexity is very restrictive and seems to exclude, e.g., the treatment of linear systems with a *full* matrix, since then even simple operations like the matrix-vector multiplication are of quadratic complexity. The survey is concluded by a discussion of this problem.

6.1 Boundary Integral Equations

A linear and homogeneous boundary value problem $Lu = 0$ in a domain $\Omega \subset \mathbb{R}^d$ can be reformulated as an integral equation of the form $\lambda u(x) = (Ku)(x) + g(x)$ for $x \in \Gamma$ with the boundary integral operator

$$(Ku)(x) := \int_\Gamma k(x,y)u(y)d\Gamma_y$$

defined on the boundary $\Gamma = \partial\Omega$. The kernel k is the fundamental solution of L or some derivative.

The advantage of the boundary integral representation is due to the fact that the domain Ω with spatial dimension d is replaced by a manifold of dimension $d-1$. Using elements of size h, the discretisation of Ω requires $O(h^{-d})$ elements, whereas Γ leads to only $n = O(h^{1-d})$ elements. In particular for exterior problems (where Ω is infinite), the integral equation is much simpler.

The disadvantage of the integral equation is caused by the fact that a discretisation of an integral operator (the so-called *boundary element method*) leads to full matrices (instead of the sparse ones for the local differential operators). For the interesting case $d = 3$, one finds that the boundary element method with dimension $n = O(h^{1-d})$ is cheaper than the standard finite element method only if the complexity is better than $O(n^{3/2})$. In particular, $O(n^2)$ complexity cannot be accepted.

This is a typical situation, where the full matrix A with its n^2 entries seems to prevent any algorithm from better complexity than $O(n^2)$. Indeed, yet the computation of the system matrix A consumes $O(n^2)$ operations where the constant may be rather large. Hence, first of all the use of the full matrix A must be avoided and replaced by a matrix (linear mapping) which can be described by (almost) $O(n)$ data. One might ask why this should be possible. The reason is that the pseudo-differential operator K has quite similar properties as standard differential operators. The latter ones can be approximated by sparse matrices depending on only $O(n)$ data.

Essentially, there are two different approaches for a realisation:

- *Matrix compression.* One can look for a special discretisation of K such that most of the entries of A are extremely small so that their replacement by zero yields an (almost) sparse matrix $\tilde{A}$. Such a discretisation can be obtained by a Galerkin approach based on suitable wavelet functions.[27]

[27]The delicate requirement is that the entries which should be suppressed must be known *before*

- *Panel clustering.* Given any discretisation of K with system matrix A, one can try to approximate A by another matrix $\tilde{A}$ which is easily describable by almost $O(n)$ data. This ensures that only almost $O(n)$ data are to be stored. Furthermore, the matrix-vector multiplication $x \mapsto \tilde{A}x$ must be performable by almost $O(n)$ operations. This is achieved by the panel clustering method.[28] The main idea is the replacement of the (smooth) kernel function k in the far-field. Different from the wavelet matrix compression, the panel clustering method is not a discretisation by itself but can be combined with any collocation or Galerkin method. Further, it is independent of the smoothness of the boundary Γ.

In both cases, one can prove that the replacement of A by $\tilde{A}$ yields an additional error which is of the same size as the discretisation error or even smaller.

6.2 General Non-Sparse Systems

Recently, L. N. Trefethen (Oxford) posed a number of maxims of which the twenty first one reads as follows:

- Is there an $O(n^{2+\varepsilon})$ algorithm[29] for solving an $n \times n$ system $Ax = b$? This is the biggest unsolved problem in numerical analysis, but nobody is working on it.[30]

Since the multiplication of a full matrix times a vector costs $O(n^2)$ operations, a sufficient condition would be that the inverse A^{-1} can be computed by an $O(n^{2+\varepsilon})$ algorithms. Unfortunately, I cannot offer such an algorithm. Instead, I would like to ask whether for a restricted (but interesting) subclass of problems, the following related question can be answered:

- Is it possible to compute a good approximate B of the inverse A^{-1} by almost $O(n)$ operations such that B requires a storage of almost $O(n)$ and such that the multiplication of B by an n-vector b costs almost $O(n)$?

At first sight, this seems impossible, since in general A^{-1} is a full matrix with n^2 entries. Indeed, for the exact inverse $B = A^{-1}$ we find only very few positive examples. However, as in the panel clustering method mentioned above, it may be possible to find an approximation $B \approx A^{-1}$ with this property.

In fact, it is possible to give a positive answer to the latter question if A is a discretisation of an elliptic operator including pseudo-differential operators. Because of the hierarchical structure of the applied matrix representation, we call the set of approximating matrices $\mathcal{H}$-matrices.[31] The precise results are as follows:

their computation. For details see, e.g., Schneider: Multiskalen- und Wavelet-Matrixkompression. Teubner, Stuttgart 1998.

[28]See, e.g., §9.7 in Hackbusch: Integral equations. ISNM 120, Birkhäuser, Basel 1995.

[29]Obviously, it is meant that ε may be any positive number. For a system with a full matrix A, which cannot be represented by less than n^2 data, $N = n^2 + n$ is the data size of the input data (A, b). Therefore, an $O(n^2) = O(N)$ complexity for solving $Ax = b$ is linear complexity!

[30]SIAM News, vol 31, No 1 (1998) page 4.

[31]Details will be in a forthcoming paper.

- the storage of the $\mathcal{H}$-matrix data is of the size $O(n \log n)$,
- the (approximate) sum of two $\mathcal{H}$-matrices costs $O(n \log n)$,
- the (approximate) product of two $\mathcal{H}$-matrices costs $O(n \log^2 n)$,
- the (approximate) product of an $\mathcal{H}$-matrix with an n-vector costs $O(n \log n)$.

Since the inverse can be obtained by multiplications (by suitable transformation matrices), also the (approximate) inversion of an $\mathcal{H}$-matrix costs $O(n \log^2 n)$ operations.

Even if one wants to perform the usual iterative techniques, often a Schur complement occurs which is of the form $S = D - BA^{-1}C$. Since the Schur complement contains the inverse matrix A^{-1}, S is usually a full matrix. Therefore, one can neither represent the matrix S nor its inverse in the standard form. Up to now, the only remedy is to know a good preconditioner for S. Then it is enough to have an efficient algorithm for the matrix-vector multiplication $x \mapsto Sx$ which can make use of the representation $S = D - BA^{-1}C$. The $\mathcal{H}$-matrix algorithm opens new possibilities, since the explicit approximate computation of $S = D - BA^{-1}C$ can be performed, provided that A, B, C, D are $\mathcal{H}$-matrices.

Wolfgang Hackbusch
Universität zu Kiel
Olshausenstr. 40
D-24098 Kiel, Germany
wh@numerik.uni-kiel.de